AFRL-RI-RS-TR-2008-175
Final Technical Report
June 2008

AGILE COMPUTING FOR AIR FORCE INFORMATION MANAGEMENT INFRASTRUCTURES

Florida Institute for Human & Machine Cognition

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2008-175 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/	/s/
ASHER SINCLAIR Work Unit Manager	JAMES W. CUSACK, Chief Information Systems Division Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JUN 2008	Final	Feb 06 – Dec 07

4. TITLE AND SUBTITLE

AGILE COMPUTING FOR AIR FORCE INFORMATION MANAGEMENT INFRASTRUCTURES

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA8750-06-2-0064

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Niranjan Suri

5d. PROJECT NUMBER
ICED

5e. TASK NUMBER
06

5f. WORK UNIT NUMBER
02

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Florida Institute for Human and Machine Cognition
40 S Alcaniz St.
Pensacola FL 32502-6008

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RISE
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2008-175

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 08-3721

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The objective of this effort was to leverage from and extend the agile computing approach and metaphor to improve Air Force Information Management infrastructures for dynamic and tactical environments. Four specific areas of research were: 1) Dynamic service instantiation, relocation and optimization, 2) Dynamic service discovery, 3) Proactive service link maintenance, and 4) Efficient data dissemination and predicate processing in dynamic networks.

15. SUBJECT TERMS
Agile computing, data dissemination, predicate processing, information management systems/infrastructures, dynamic and tactical environments

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Asher Sinclair
U	U	U	UU	26	19b. TELEPHONE NUMBER (*Include area code*) N/A

Table of Contents

List of Figures

1. Introduction

The objective of this effort was to leverage from and extend the agile computing approach and metaphor to improve Air Force information management infrastructures for dynamic and tactical environments. Four specific areas of research were: (a) Dynamic service instantiation, relocation, and optimization, (b) Dynamic service discovery (c) Proactive service link maintenance, and (d) Efficient data dissemination and predicate processing in dynamic networks.

This document is the final technical report for the project, which began in March 2006 and ended in December 2007. Section 2 presents background information on the notion of Agile Computing. Section 3 presents the overall architecture for the agile computing middleware. Section 4 presents details about each of the four specific research areas targeted by this effort. Section 5 presents some potential future directions to continue this research. Finally, the appendix contains papers that resulted from the research sponsored as part of this project.

2. Background

Operational and tactical military environments are composed of mobile nodes and dynamic situations resulting in a high degree of uncertainty. Communication links are often based on ad-hoc networks resulting in intermittent reachability, variable latency, and low bandwidth. As nodes enter and leave the environment, the set of available data sources and services within the Community of Interest change continuously. Information management infrastructures must be designed to operate effectively in such environments in order to support the vision for net-centric operations.

2.1. Overview of Agile Computing

Agile computing is an innovative metaphor for distributed computing systems and prescribes a new approach to their design and implementation. This report describes the overall agile computing metaphor as well as one concrete realization through a middleware infrastructure.

Agile computing may be defined as opportunistically discovering, manipulating, and exploiting available computing and communication resources in order to improve capability, performance, efficiency, fault-tolerance, and survivability. The term **agile** is used to highlight the desire to both quickly react to changes in the environment as well as to take advantage of transient resources only available for short periods of time. Agile computing thrives in the presence of highly dynamic environments and resources, where nodes are constantly being added, removed, and moved, resulting in intermittent availability of resources and changes in network reachability, bandwidth, and latency.

From a high-level perspective, the goal of agile computing is to facilitate resource sharing among distributed computing systems. At this broad level of description, agile computing overlaps with several other areas of research including distributed processing, peer-to-peer resource sharing, grid computing, and cluster computing. The following factors constrain the goals of agile computing and differentiate it from the other areas of research.

Transient Resources: The resources in the target environment for agile computing are expected to be highly transient. Indeed, one of the performance metrics for agile computing is defined as a function of the minimum length of time that a resource must be available in order to be utilized productively. The expectation is to support environments where resources are available on the order of seconds or minutes, as opposed to hours, days, or longer. Therefore, agile computing differs from grid computing and cluster computing, which targets environments where resources are more stable.

Limited Communications: The networks used to interconnect resources in the target environment are expected to be wireless and ad-hoc. The implication for agile computing is that the middleware must be able to support and operate in low-bandwidth, high and variable latency, and unreliable networks. Again, this differentiates agile computing from grid computing and cluster computing, where the network links tend to be reliable and high in performance.

Opportunistic Resource Exploitation: Another goal of agile computing is to take advantage of unexpected resources that happen to be available. In particular, the goal is to take advantage of resources that were not originally intended to be tasked, but happen to have spare capacity for transient periods of time. Some peer-to-peer systems provide the same capabilities. Agile computing extends this capability to also be proactive and manipulate the resources in the environment to satisfy the requirements. Manipulation can include physically moving resources (for example, robots or other autonomous vehicles) in order to provide communications or processing capabilities where required. These additional resources may be uncommitted resources assigned for use by the middleware or other resources which can be manipulated without interfering with their original task assignments. This aspect differentiates agile computing from other peer-to-peer systems as well as from grid computing and cluster computing.

The following subsection describes a scenario that can benefit from the concepts of agile computing. They also describe the target environments and conditions under which the agile computing middleware needs to operate.

2.2. Agile Computing for Military Information Networks

The concept of agile computing and the agile computing middleware were originally conceived to address the challenges presented by military information networks. Figure 1 below shows one conceptual view of a tactical military environment.

From the perspective of agile computing, there are several interesting characteristics in such an environment. As observed in the figure, the environment is expected to be chaotic, with a wide range of platforms operating concurrently. Platforms include satellites, manned aircraft, unammed aircraft (UAVs), tanks and other manned ground vehicles, robotic vehicles, ships at sea, unmanned underwater vehicles (UUVs), unattended ground sensors (UGSs), and dismounted solders. All of these platforms are interconnected through a variety of networking mechanisms ranging from acoustic modems to RF (Radio Frequency) modems to wireless ad-hoc networks to long-range wireless links. For example, a set of ground sensors may establish an

ad-hoc network among the sensors, with one of the nodes acting as a gateway that interconnects with the rest of the network hierarchy through a modified 802.11 interface.

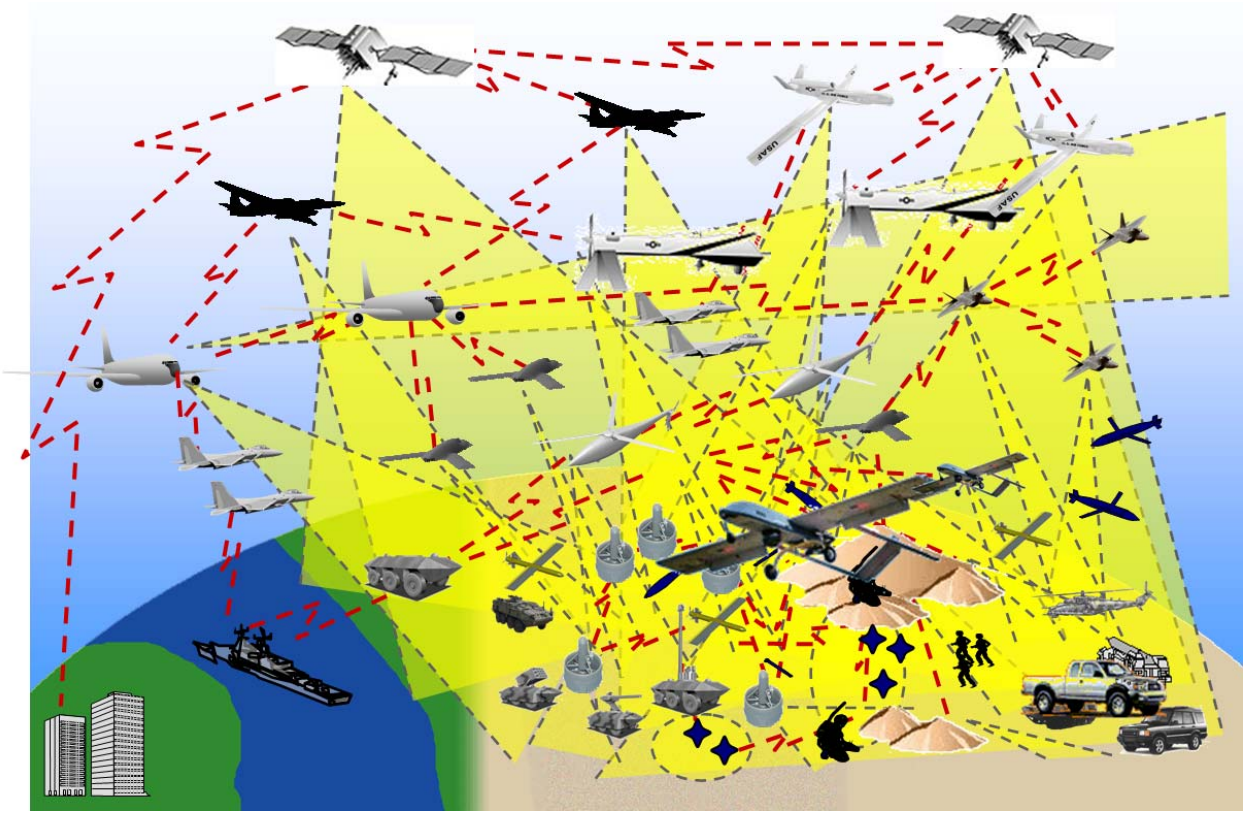

Figure 1: A Conceptual View of a Tactical Military Environment

(Courtesy of Dave Gunning at DARPA)

The platforms (henceforth referred to as resources) vary greatly in their range of capabilities and in their roles. For example, some resources, such as ground sensors, are battery powered and have severe constraints on power utilization due to the requirement that they continue to operate for some desired duration of time after deployment. Resources vary in their processing capabilities, memory and storage capabilities, communication capabilities, and display capabilities. Some resources have dedicated roles (for example, the fire control computer in a tank) while others are general purpose computers (rack mounted servers in a HMMVW that provide services such as a GIS database or a directory service).

New resources may be dynamically added or removed from the environment. For example, a new set of sensors may be dropped into the battlefield environment thereby increasing the number of available resources. Resource attrition is to be expected due to enemy attacks and usage of consumable resources (such as a ground sensor running out of battery power). Also, since many resources such as aircraft, UAVs, and other vehicles are mobile, they may enter and leave the environment at arbitrary times. From the perspective of agile computing, a resource enters the environment when communications can be established to the resource and the resource leaves when communications is no longer possible. Note that temporary disconnection of a

resource from the network due to unreliable or disrupted communications is also possible and may need to be differentiated from a resource entering and leaving the environment.

In a battlefield environment, there are typically multiple parallel activities overlapping in space and time. For example, while the Army may have forces operating on the ground, the Air Force may have aircraft operating in the air, either in support of the Army or on other parallel missions. In addition to multiple forces, there are likely to be coalition forces operating in cooperation. In urban environments, other groups such as the Red Cross may be active as well. These different groups bring different sets of resources to bear within the same environment. From the perspective of a mission, the result of such parallel activities is that other unexpected resources might become available during the course of the mission. For example, an Army soldier wanting to get an image of an area on the other side of a hill might be able to tap into a camera on an Air Force plane that just happens to be over the area of interest. Being able to opportunistically take advantage of such resources can help provide new capabilities as well as offset the attrition of resources that is almost certain to occur.

Given the coalition nature of the majority of military operations, resources in the battlefield are typically owned by different stakeholders. Even in single-nation operations, there are likely to be different services or branches of the military involved. The implication is that there are effectively multiple chains of command and administrative domains leading to constraints on resource sharing between the stakeholders. These different stakeholders might be collaborating to perform a single mission or may be operating in parallel as discussed above. In the former case, resource sharing agreements would already be in place while in the latter case, resource sharing would have to be spontaneous. To facilitate such sharing, the system should allow constraints on resource utilization across domains to be specified ahead of time and let resources be opportunistically tasked within those constraints.

The communications infrastructure is wireless and consists of a number of different technologies – microwave, 802.11, and software programmable radios such as JTRS. The networking technology is likely to be ad-hoc in order to not require a significant effort to deploy and setup a communications infrastructure. Such wireless ad-hoc networks provide low-bandwidth, high-latency links. Moreover, these links are unreliable as devices are moving and going in and out of communications range with respect to each other. Under some circumstances, it is also important to minimize data transmission in order to conserve battery power or to operate in a clandestine manner.

There are a wide range of devices operating within the battlefield environment, ranging from unattended sensors with small embedded computers to platforms with custom computer systems to standard rack-mounted servers in HMMWVs. These devices vary greatly in their CPU architectures, hardware capabilities, available power, and size. A UAV or a tank may have computers with standard x86 compatible processors, plenty of memory and storage, a high-bandwidth network link, and no constraint in terms of battery power. On the other hand, an embedded computer on a sensor may have a custom processor, limited memory, no secondary storage, a low-bandwidth network link, and be constrained by the limited battery capacity.

Within the battlefield environment, the producers of information are typically sensors (unattended ground sensors, cameras on robots and UAVs, etc.) and soldiers operating in the

field. The consumers of information range from dismounted soldiers to pilots to commanders at various levels of the hierarchy. Some of the endpoints involved in this information flow are weak resources (in terms of computational power, battery power, storage, and communications). The power constrained sensors and the disadvantaged users need to take advantage of other intermediate resources in battlefield environment that have the necessary capacity.

2.3. Target Environment Characteristics

The scenario described above is useful to characterize the target environment for agile computing. The computing resources in the environment vary over time along six dimensions: their architecture (CPU and operating system), their capabilities (processing, memory, storage, communications, and power), their ownership, their role, their location, and their current load. In addition, as their location within the environment change, the reachability of the resources may change over time. In terms of communications, the networks are wireless and ad-hoc, resulting in continuously changing reachability, bandwidth, and latency. The environment may be viewed as a system of systems, incorporating distributed sensor networks, teams of autonomous vehicles, groups of human operators (with associated resources), and high-capacity back-end computing systems.

2.4. Agile Computing Middleware Goals

2.4.1. Providing and Maintaining Communications Links: Systems within the environment need to communicate and exchange information for a variety of reasons. A sensor may need to be queried in order to obtain data. A robot may need to be driven by a human operator. A team member may need to retain connectivity with other members in order to be aware of their positions. A database may need to be queried to retrieve relevant information. The available communications resources need to be effectively allocated to competing demands based on policies. For example, an operator driving a robot may need to be provided a low-latency link with a limit on the minimum bandwidth available. Satisfying these requirements may require manipulation of the environment (for example, moving a robot to act as a communications relay).

Resource manipulation may be integrated in two different modes. In the first approach, manipulation is invoked only when the resource requirements cannot be satisfied by the currently available resources in their current configuration. An example would be moving a communications relay into place when there is no existing link or the link does not provide the necessary characteristics. In the second approach, resource manipulation is more tightly integrated into resource allocation. Being able to manipulate the environment provides an extra degree of freedom beyond simply allocating among available resources. An example would be moving a communications relay even if there was a communication path already available, if moving the resource to a new location (at a cost) provides a larger savings in overall communication costs.

2.4.2. Supporting Distributed Processing: Given the wide range of devices and capabilities in the target environment, computational power is not evenly distributed. Users working with small PDAs or tablet computers are limited in their processing capabilities and constrained by the

limited battery power. The middleware needs to support such disadvantaged users to exploit other computing resources in the environment. For example, a soldier on the ground may be using a tablet computer to run a data analysis algorithm. If a UAV or a HMMVW with a more powerful computer were to come into communications range with the soldier, the middleware should allow the tablet computer to offload the processing to the more powerful resource while the resource is available. If the resource were to go away before the processing is completed, the middleware should be able to migrate the computation onto another resource or back to the original tablet computer.

One of the challenges that needs to be addressed by the middleware is the wide variety of resource types. The computing resources are expected to have different CPU architectures and different operating systems. Therefore, the middleware must be able to support migrating computations across different platforms. Moreover, this migration should be transparent to the computations themselves, which requires support for process migration and redirection of resources that the computations need to access.

2.4.3. Embedding Processing Along the Data Path: Data dissemination is one common application in tactical military environments. Data may be gathered by human operators or by sensors that are distributed in the battlefield environment. Applications normally use a middleware publish-subscribe system to deliver data from the data producers to consumers. Capabilities such as data fusion, hierarchical data distribution, and policy enforcement require that the data be processed before delivering the data to consumers. For example, data fusion may require that two different sensors transmit data to a processing component that transforms the incoming data and produces a new data stream for one or more consumers. Policy enforcement may require that sensor data be transformed before being delivered to a consumer. Similarly, hierarchical data distribution may require that a process generate a subset of an existing data stream in order to service a new client request. In all three of these examples, processes need to operate upon the data produced before the data can be delivered to the consumers.

In a tactical military environment, the data consumers and data producers may be underpowered resources. For example, small embedded sensor devices may not have much processing power or storage capabilities. Likewise, dismounted soldiers carrying PDAs or other small computers are constrained by the capabilities of their devices. The implication of the endpoints being weak resources is that any data processing must be accomplished on other intermediate resources in the environment. The agile computing middleware provides the underlying capabilities to opportunistically discover and task resources in the environment that can support the data processing requirements of the publish-subscribe system.

3. Agile Computing Architecture

This section presents the overall architecture for the agile computing middleware. This architecture is one instance of the agile computing model presented in the previous section. Fundamentally, the architecture supports the coordination, communication, distribution, and execution of processes on opportunistically discovered nodes.

The overall architecture for the agile computing middleware consists of a kernel component, a coordination component, a visualization component, and APIs for applications to interact with the middleware. Figure 2 shows a typical configuration with a number of services and clients.

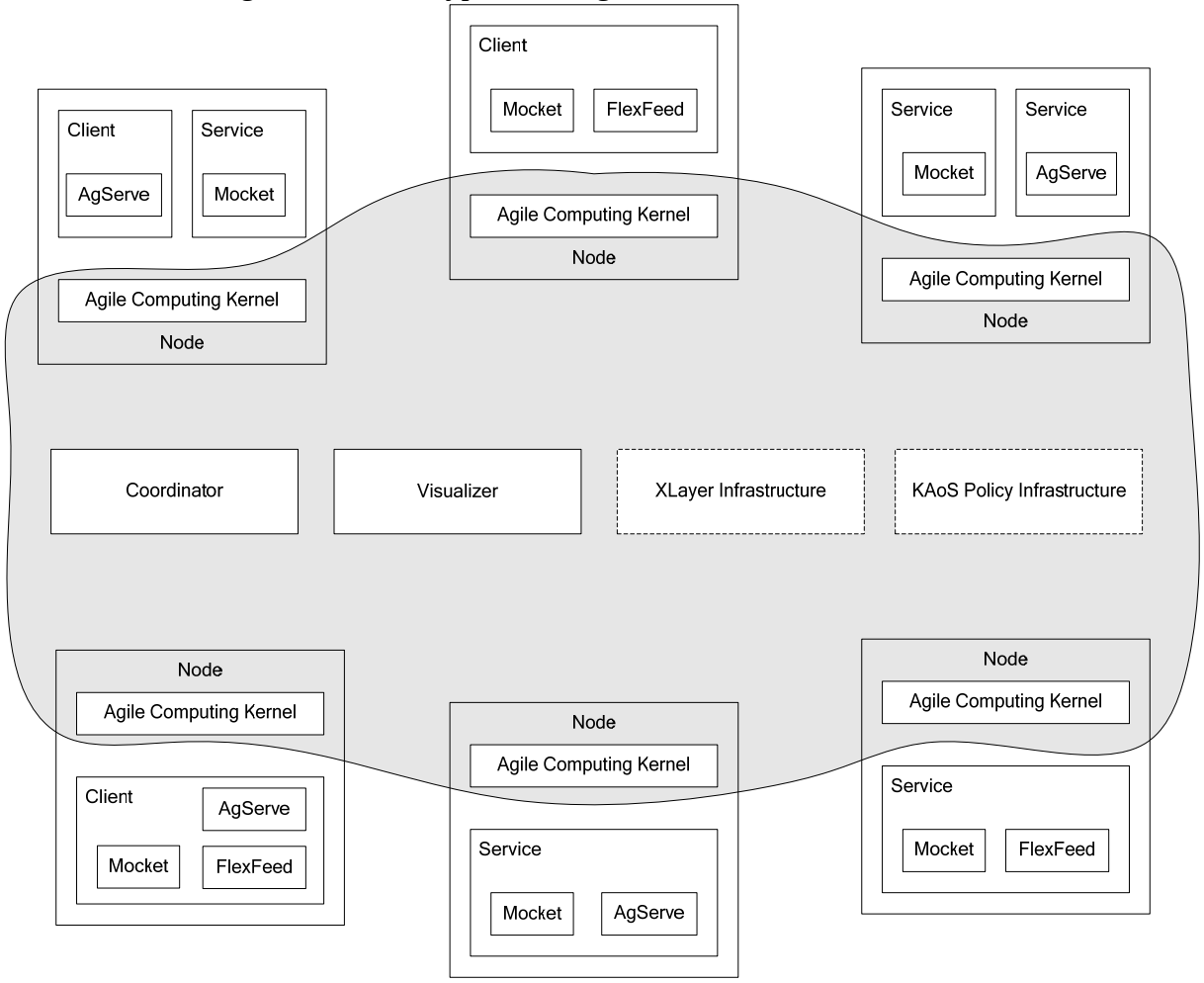

Figure 2: Overall Architecture for the Agile Computing Middleware

Each node executes an instance of the Agile Computing Kernel (ACK). Nodes may be running client applications, services, or any combination. Client applications interact with the middleware using one of three APIs: **Mockets**, **AgServe**, and/or **FlexFeed**.

In addition, the **Coordinator** and the **Visualizaer** are middleware components that work with the ACKs running on each node. The Coordinator, while shown in a single box, may be in fact realized using three different approaches: a centralized approached with a single coordinator, a zone-based approach with a centralized coordinator per zone (with the zones organized into a hierarchy), and a fully distributed approach, where coordination is performed by each kernel communicating with other kernels directly. These approaches are further discussed in section 2.2 below.

The other two components shown in the dashed boxes, **XLayer Infrastructure** and **KAoS Policy Infrastructure**, are components used by the middleware but are separate frameworks. They are not a part of the agile computing middleware.

3.1. Agile Computing Kernel

The kernel contains a number of components that realize various functions of the middleware. Figure 3 shows the components that the kernel has in the normal configuration. These components are briefly described below. The more significant components are described in detail later.

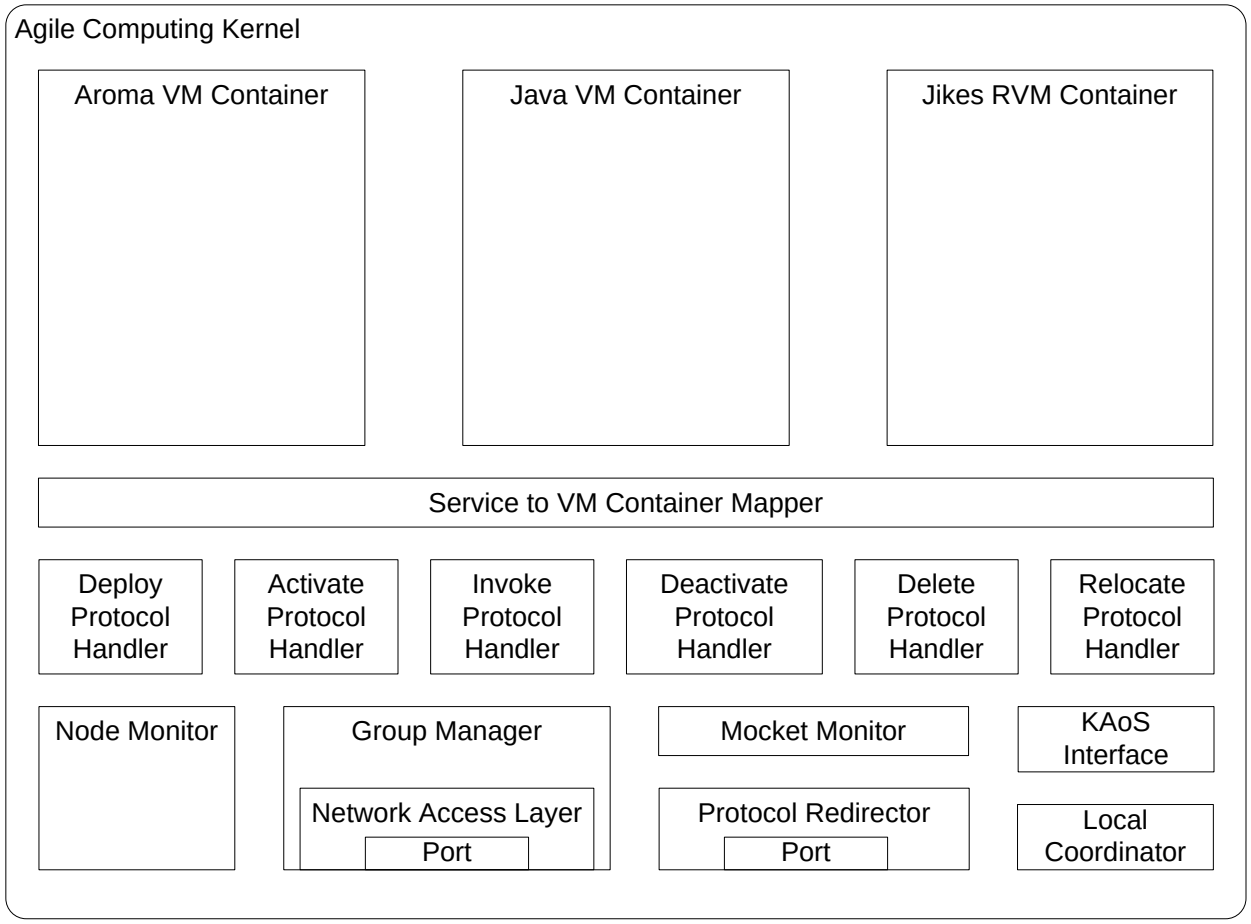

Figure 3: The Agile Computing Kernel

Node Monitor: Monitors the resource utilization at the local node, including CPU, memory, storage, and network.

Group Manager: Handles discovery of other nodes and grouping of nodes into different sets. Also handles propagation of resource information from the local kernel to other kernels and searching for nodes or services at other nodes.

Mocket Monitor: Monitors the status of mockets being used on the local node by clients or services, including open connections, failed connections, lost connections, and statistics about the connection.

KAoS Interface: Connects with the KAoS Policy and Domain Services infrastructure to obtain policies that can be used to control the behavior of the kernel and other components (e.g., limiting the bandwidth utilized by a mocket connection).

Local Coordinator: Handles coordination for resource allocation and service life-cycle management. It works in conjunction with other local coordinators or centralized or zone-based coordinators to perform resource allocation, service deployment, service invocation, and service migration decisions. The coordination mechanism is discussed further in section 2.2.

Protocol Redirector: Handles incoming connections to the kernel from clients or from other kernels and directs the connection to the appropriate protocol handler.

Protocol Handlers: Realize particular protocols that are used when performing functions such as deploying a new service, activating a service, invoking a service, and migrating a service, among others.

VM Containers: Execute Java VMs which, in turn, execute services as part of the kernel. The internals of the VM Containers are discussed below. There are currently three variations – one for the Aroma VM, one for Sun's Java VM, and one for the Jikes RVM.

VM Container Mapper: Keeps track of the VM Container that is executing each service, which allows the protocol handlers to direct incoming requests to the right VM Container.

3.2. Coordinator

The coordinator is the logical entity that manages the overall behavior of the agile computing middleware. The coordinator monitors node and resource availability as well as network connectivity and bandwidth. Client requests are received by the coordinator, which handles allocation of resources. The coordinator also performs proactive manipulation of nodes, such as moving a node to act as a relay in order to restore a lost communications link. Coordination is a continuous process as the resource allocation needs to adapt to changes in the environment.

The coordinator may be realized using either a centralized approach, a zone-based approach, or a fully distributed approach. Figures 4, 5, and 6 show the three different approaches.

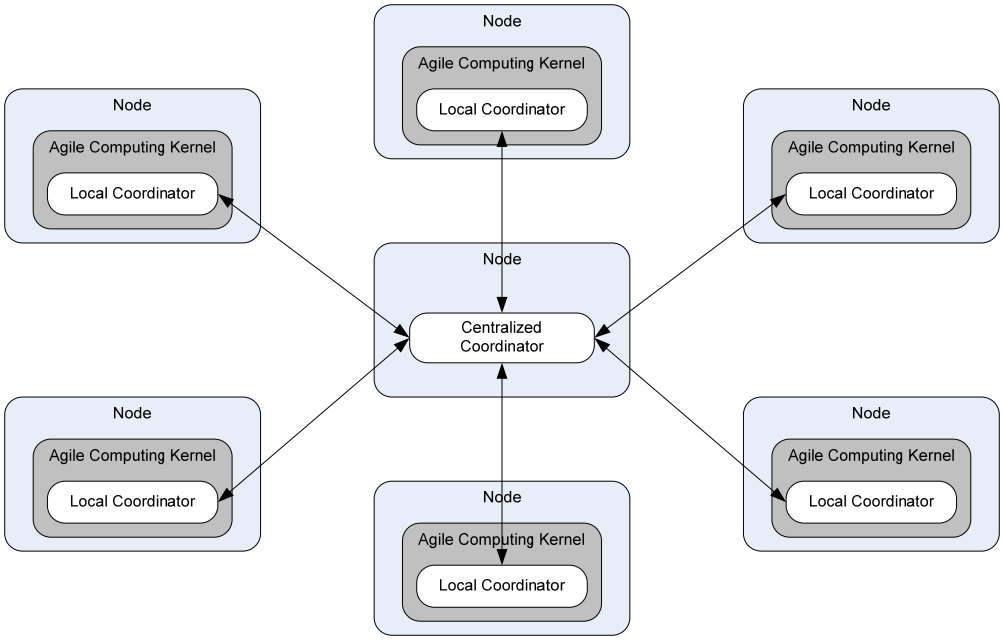

Figure 4: Centralized Coordination Approach

In the centralized approach, a single node behaves as the coordinator at any given point in time. This coordinator communicates with the local coordinator at each of the nodes. Requests from clients are sent to the coordinator, which in turn issues commands to the other nodes. If the coordinator (or the node) fails, then a new coordinator can be selected through an election process.

Like any other centralized approach, the centralized coordinator is the simplest, but is not scalable and introduces a bottleneck as well as a single point of failure.

In the zone-based coordination approach, nodes are grouped into zones, each of which has the equivalent of a centralized coordinator. Zones are typically created based on network proximity (which, in wireless environments, also implies physical proximity). One of the nodes in a zone is elected to be the zone coordinator. This zone coordinator interacts with all the other local coordinators within the zone (much like the centralized approach) and also with other zone coordinators.

Two structural arrangements are possible with the zone-based approach – peer-to-peer and hierarchical. Figure 5 shows a fully connected peer-to-peer arrangement between the zones, although it is not necessary for all of the zone coordinators to be in direct communication with each other.

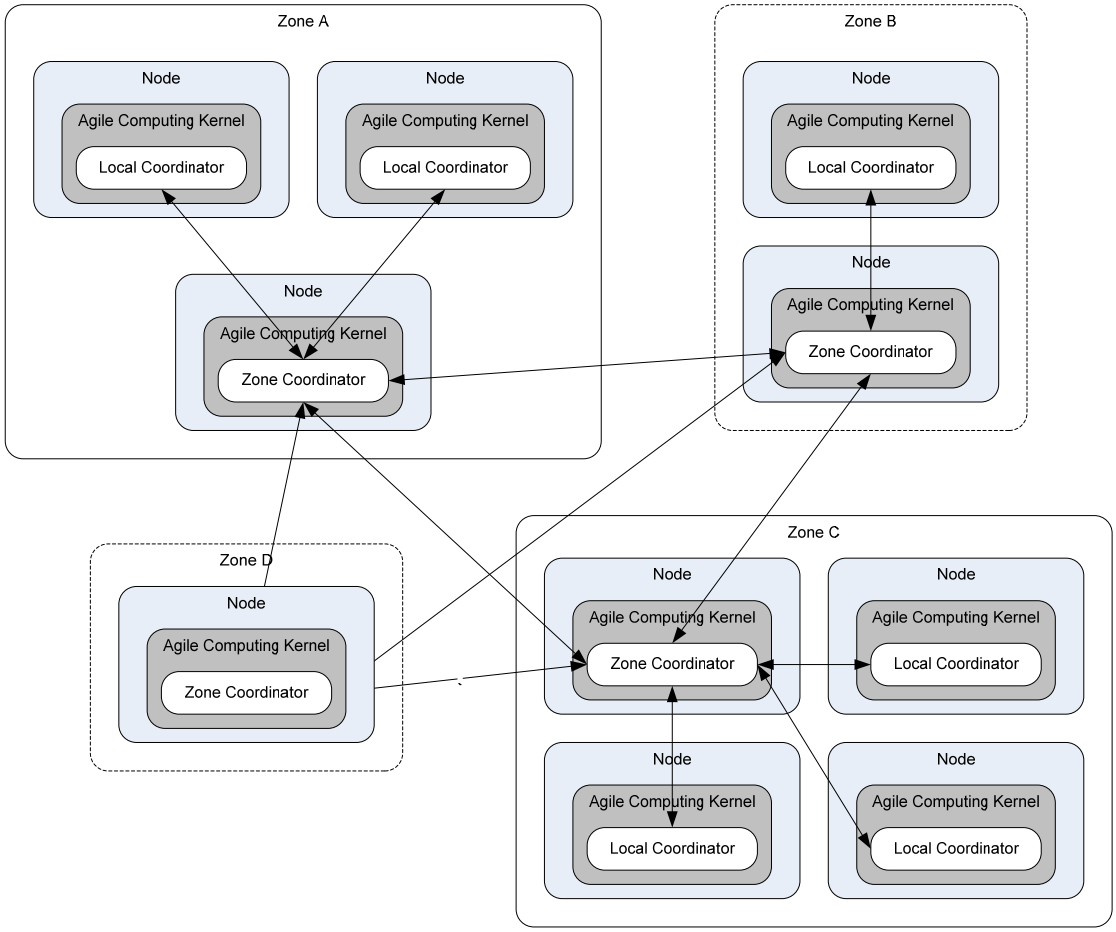

Figure 5: Zone-based Coordination Approach

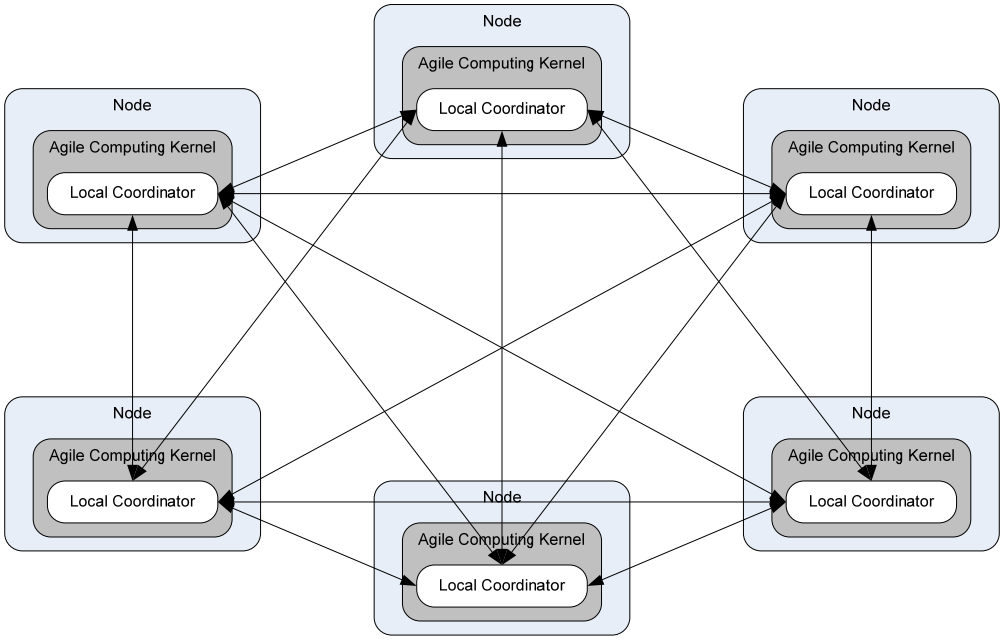

Figure 6: Distributed Coordination Approach

The last possibility is a fully distributed coordination approach, as shown in Figure 6. In this approach, there is no centralized or partially centralized coordinator at all. Each of the local coordinators directly communicates with other local coordinators as needed. Again, Figure 6 shows a fully-connected arrangement, but that is not a requirement. The fully distributed coordination approach does not have a single point of failure but like most distributed algorithms, it is the most complicated approach.

In addition to the three different approaches, a number of different coordination algorithms are possible, based on the context and the problem to which the agile computing middleware is applied. The first version of the coordinator has been designed for the FlexFeed framework – which applies the notion of agile computing for sensor information feeds. The FlexFeed coordinator uses the centralized approach and handles path optimization, efficient data distribution, and policy enforcement. FlexFeed has been used both in the context of the DARPA Control of Agent-based Systems (CoABS) and the U.S. Army Future Combat Systems (FCS) research programs.

The overall goals and desired behavior of the coordinator can also be regulated via policies. Policies do not specify the coordination strategy, but rather runtime constraints on the strategy. For example, policies can be used to specify that only 50% of a node's CPU should be used, or that a node with less than 1 hour of battery life should not be exploited. Policies are specified using the KAoS framework, which allow the administrators or maintainers of the system to dynamically change behavior at runtime.

4. Focused Research Areas

4.1. AgServe – Dynamic Service Instantiation, Relocation, and Optimization

AgServe is the service-oriented architecture that was built and integrated into the Agile Computing Middleware as part of this effort.

We began by investigating existing web services architecture and technology. This survey was done prior to designing and implementing APIs for a service oriented architecture on agile computing. The goal was to examine current standards and available COTS and open source products to determine their adequacy and usability for agile computing.

OASIS and the W3C are the primary committees responsible for the architecture and standardization of web services. Over the last few years, three primary technologies have emerged as worldwide standards that make up the core of today's web services technology: Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI).

SOAP provides a standard packaging structure for transporting XML documents over a variety of standard Internet technologies, including SMTP, HTTP, and FTP. By having a standard transport mechanism, heterogeneous clients and servers can become interoperable.

WSDL is an XML technology that describes the interface of a web service in a standardized way. WSDL standardized how a web service represents the input and output parameters of an invocation externally, the function's structure, the nature of the invocation (in only, in/out, etc.) and the service's protocol binding. WSDL allows disparate clients to programmatically understand how to interact with a web service.

UDDI provides a registry of web services for advertisement, discovery, and integration purposes. The web service information is published using this protocol.

Among the different toolkits and APIs that facilitate the creation and deployment of web services, Apache Axis2 is the most promising one. Apache Axis2 is an open source, Java and XML based Web Services framework consisting of an implementation of the SOAP server and various utilities and API's for generating and deploying Web service applications. Besides the Java version, a C++ implementation is also available. When a Web service is deployed using Axis2, it will generate a WSDL file automatically that can be used to generate the Web service client stubs for different programming languages (e.g. Java, C#, etc.).

Web services may suffer from poor performance compared to other distributed computing architectures such as RMI, CORBA, or DCOM. This is a common trade-off when choosing text-based formats. XML explicitly does not count either conciseness of encoding or efficiency of parsing among its design goals. Binary representations such as SOAP MTOM promise to improve the wire efficiency of XML messaging.

Based on our research, we have decided to implement the following approach for agile computing.

We have considered two distinct models for synchronous service invocation. The first option involved the full specification of the service request and service response via SOAP messages to be exchanged between the client and the service. This alternative follows the conventional mechanism proposed for method invocation in web-services standards but has some drawbacks in tactical environments.

The second model considered for synchronous service invocation was based on the remote procedure call mechanism provided as part of SUN's Java™ language. The RMI-based method is highly efficient and compact (in terms of message complexity).

In Java RMI, a client node can invoke a method at a server node (service) through a local service-stub that provides all exposed services methods. Request and response follow a proprietary format that includes all objects and arguments as serialized Java objects. Primitive types are converted to complex objects prior to serialization at the client side.

The downside of an RMI-like strategy for tactical environments lies in the fact that the service request and response methods provide no intelligible information that could be exploited by the multi-hop nature of such networks.

After considering both alternatives, we have opted for utilizing a hybrid approach for service invocation, utilizing a SOAP envelop with limited information about the contained RMI-like serialized request. The SOAP envelop will contain the name of the class, method and arguments (as well as primitive values) for the method. All complex objects associated with the request or response will be serialized and wrapped by nested elements also containing the associated class names.

The proposed mix-approach for method invocation allows complex objects to be easily (and efficiently) exchanged between nodes and still provides some structure and basic information about the objects, methods and arguments involves, with can utilized by intermediate nodes in the network.

The AgServe implementation, service migration, and experimental results are described in two papers included in the appendix:

- Towards an Agile Computing Approach to Dynamic and Adaptive Service-Oriented Architectures
- An Adaptive and Efficient Peer-to-Peer Service-oriented Architecture for MANET Environments with Agile Computing

4.2. Group Manager – Dynamic Service Discovery

The Group Manager is an application-level component that supports resource and service discovery. It enables the agile and opportunistic exploitation of resources by optimizing queries to find nodes in network proximity and/or nodes that are resource rich or have excess capacity.

The Group Manager supports proactive advertisement, reactive search, or a combination of the two. The search is realized using a Gnutella-like or probabilistic search mechanism. In addition, the radius (in terms of network hops) of the advertisement or search can be controlled on a per request basis, providing a powerful mechanism to control how strongly or weakly a service may be advertised and how far a search request may travel. In the simplest case, distance is defined as the number of hops in a MANET environment, but can be a more application relevant parameter such as bandwidth or latency.

Propagation of the advertisement and search messages occurs via one of three mechanisms – UDP broadcast (the simplest case), UDP multicast, or via the XLayer Framework that provides bandwidth-efficient flooding. In addition, tunneling via TCP supports bridging multiple networks. In all of these four cases, each node may selectively rebroadcast an incoming message to provide control over the radius.

These capabilities enable applications to make tradeoffs between discoverability, bandwidth, and latency. Proactive advertisement uses more bandwidth but reduces the latency when a client needs to find a service and vice-versa. On another dimension, a service that is widely present in a network does not need to advertise strongly (or, consequently, a client looking for such a service does not need to search widely) as opposed to services that are scarce.

Groups may be used to partition network nodes into different sets thereby restricting advertisements and queries. The Group Manager provides support for two different group types: peer groups and managed groups. Peer groups are completely decentralized: they do not have an owner or manager, but instead maintain node membership independently from the perspective of each node. This design choice also implies that there is no attempt at maintaining a consistent group view for all nodes that are members of a peer group. In addition, no special mechanism is required in order to join a peer group. Nodes can simply query or register resources and services in the context of a specific peer group and will implicitly be treated by the Group Manager as members of the same peer group.

The decentralized and dynamic nature of peer groups make them very well suited for resource and service sharing in MANET environments. However, for additional flexibility, the Group Manager also supports centralized resource and service management by means of managed groups. A managed group is created by a particular node and is owned by that node. Other nodes need to explicitly join a managed group by sending a membership request to the group owner node.

Access to groups (of both peer and managed types) may optionally be restricted using a password, thereby preventing nodes that do not possess the necessary authorization credentials from joining a specific group.

The Group Manager API has been designed to be extremely simple and generic, facilitating its use in a wide range of applications. For example, the ACI Kernel uses the Group Manager to propagate node resource information, the Service Manager uses it to publish services in XML and search for services using XPath queries, and FlexFeed uses the Group Manager to find data sources for subscribers. Versions of the Group Manager are available for both Java and C++, and

operating at the application layer facilitates easy, piecewise integration into existing applications. All of the above features combine together to make the Group Manager well suited to MANET environments.

The Group Manager is further described in two papers included in the appendix:

- Resource and Service Discovery in Wireless Ad-Hoc Networks with Agile Computing
- An Adaptive and Efficient Peer-to-Peer Service-oriented Architecture for MANET Environments with Agile Computing

4.3. Mockets – Proactive Service Link Maintenance

Mockets (for "mobile sockets") is a comprehensive communications library for applications. The design and implementation of Mockets was motivated by the needs of tactical military information networks, which are typically wireless and ad-hoc with low bandwidth, intermittent connectivity, and variable latency. The initial implementation of Mockets was completed for use by the Army Research Laboratory as part of the Warrior's Edge initiative of the Horizontal Fusion Portfolio's Quantum Leap demonstrations.

Mockets addresses specific challenges including the need to operate on a mobile ad-hoc network (where TCP does not perform optimally), provides a mechanism to detect connection loss, allows applications to monitor network performance, provides flexible buffering, and supports policy-based control over application bandwidth utilization.

Mockets has been designed to provide the following five capabilities:

1) Application-level implementation of the communications library in order to provide flexibility, ease of distribution, and better integration between the application and the communications layer.
2) A TCP-style reliable, stream-oriented service that is designed to operate on wireless ad-hoc networks thereby making it easy to port existing applications to the ad-hoc environment.
3) A message-oriented service that provides enhanced capabilities such as message tagging and replacement, different classes of service (reliable/unreliable combined with sequenced/ unsequenced), and prioritization.
4) Transparent mobility of communication endpoints from one host to another in order to support migration of live processes with active network connections.
5) Interface to a policy management system in order to allow dynamic, external control over communications resources used by applications.

The result is a flexible user-level communications library with implementations in Java, C++, and C#. The performance of Mockets is equal to or better than TCP for the type of networks that were targeted, while providing the additional desired capabilities.

Mockets is further described in the following three papers included in the appendix:

- Mockets: A Novel Message-Oriented Communications Middleware for the Wireless Internet
- Network Conditions Monitoring in the Mockets Communications Framework
- Session Mobility in the Mockets Communication Middleware

4.4. FlexFeed – Efficient Data Dissemination and Predicate Processing

FlexFeed is a publish/subscribe communications framework for dynamic in-stream data processing in mobile ad-hoc network environments under policy and resource constraints. The framework uses mobile agents as data-aware processing elements to better customize multicast trees and allocate in-stream data processing capabilities in the network. In-stream data processing relies on taking advantage of the multi-hop nature of network paths in ad-hoc networks to appropriately allocate data processing elements for some optimization criteria.

In the agile computing middleware, FlexFeed is the API that application utilize for publish-subscribe oriented communications. In the context of data-aware publish-subscribe systems, it opportunistically allocates computational resources in the network while taking into consideration load, connectivity, and bandwidth availability in order to minimize overall costs for data processing and distribution of multiple concurrent data feeds.

In order to provide decentralized mechanisms for high level policy definition, deconfliction and distribution, the FlexFeed framework has been integrated with KAoS policy services as its default framework but other approaches could be straightforwardly adapted. Upon policy distribution, the FlexFeed framework is responsible for providing on-demand deployment and activation of policy enforcers.

In FlexFeed, policies can be used to regulate local (and global) resource utilization of concurrent data feeds, as well as to regulate the context-dependent release of information between nodes. FlexFeed supports the deployment of customized data filters at run-time that can be used as data-aware policy enforcers for specialized data types. For example, details of images being transmitted by a particular camera sensor can be transparently downgraded for clients not authorized to access the full resolution video.

Policies can also be used to regulate and constrain the autonomous behavior of the framework, providing bounds for self-adjustments to operation tempo and to the proactive manipulation of resources. For example, policies can specify the conditions under which resources can be used or moved in order to restore communication loss.

While other aspects of policy management are performed by KAoS, the enforcement of policies is autonomously handled by FlexFeed. The framework will opportunistically allocate (and monitor) the necessary resources for policy enforcement. When resources available are insufficient for the policy requirements, the framework reports the issue to the policy infrastructure, requesting assistance.

FlexFeed is further described in the following two papers included in the appendix:

- A Mobile Agent-based Communications Middleware for Data Streaming in the Battlefield
- Policy-based Bandwidth Management for Tactical Networks with the Agile Computing Middleware

5. Future Directions

The agile computing middleware provides numerous capabilities that are well suited for tactical military environments. As the middleware is utilized in different contexts and for different experiments, new requirements are identified, both in terms of useful extensions to existing components and for entirely new components. Some of these are described below:

5.1. Enhancements to Mockets

5.1.1. Support for multiple, simultaneous network streams

The Mockets communications library supports the notion of network mobility - where a node migrates from one network to another thereby changing the IP address that has been assigned to the node. If applications are using Mockets, the change is transparently handled by the Mocket endpoint.

An enhancement to this capability would be to support multiple IP addresses (and multiple interfaces) transparently as they become available and unavailable. For example, consider a laptop that has a wireless interface that is active. If the throughput is insufficient, a user might decide to plug the laptop into the wired Ethernet network. However, existing applications that have open network connections would not switch to using the new link, thereby still experiencing limited throughput until connections are stopped and restarted. This capability would allow mockets to not only switch transparently to the Ethernet network, but to use both in parallel. The state estimation features in Mockets could be used to detect the quality of the network links and split the traffic accordingly (or as specified through an application-level or system-level policy).

5.1.2. Adaptive congestion control

Existing congestion control algorithms do not work well in wireless environments since they often conflate temporary wireless disconnection with network congestion. Mockets needs to be enhanced with new congestion control algorithms that are designed to work well in wireless networking environments.

5.1.3. Reliable-until-replaced Semantics

Mockets provides the notion of message replacement – where a new message can replace an outdated message in order to reduce the amount of network traffic. However, for any messages or fragments that have already been transmitted but not yet acknowledged, message replacement requires the generation of a control packet indicating that the previously transmitted messages have been cancelled. This allows the receiver to discard and not wait for any missing fragments in the prior messages.

The reliable-until-replaced semantics provides a similar capability to message replacement, but without the overhead of the control packets to cancel messages. This approach would be more efficient, but would only work in situations where the application does not want to selectively

cancel individual packets. The new semantics are well suited for situations such as position updates, where a new update invalidates all previous updates. The original message replacement approach is still required to support situations where a new update only invalidates a subset of previous updates.

5.2. Porting JBI to AgServe

The AgServe service-oriented architecture has been designed and implemented as part of this effort to support dynamic SOAs that opportunistically exploit nodes to run services. The next step would be to port some of the JBI components and package them as services on top of AgServe and experiment with them to measure performance improvements in tactical environments.

5.3. Peer-to-Peer Information Dissemination

One of the new capabilities that has been identified as a requirement for the agile computing middleware is a peer-to-peer information dissemination service. The goal for this service is to integrate the notion of groups from the Group Manager and extend the point-to-point message transmission capability of Mockets to support point-to-multipoint dissemination of information. In addition, the dissemination service will support store and forward to handle network disconnections and partitioning. The dissemination service will also need to adapt its behavior to support multiple patterns of publishers and subscribers, including one-to-one, one-to-most, most-to-most, most-to-one, one-to-few, few-to-few, few-to-one, few-to-most, and most-to-few.

The dissemination service could provide a more robust and efficient means for dissemination in tactical environments and could be the foundation for a peer-to-peer version of JBI.

5.4. Integrating Learning to Support Adaptation Over Time

Several aspects of the behavior of the agile computing middleware and Mockets, Group Manager, FlexFeed, AgServe, (and the Dissemination Service) can benefit from dynamic adaptation to the environment. Such dynamic adaptation requires that the behavior of various aspects of the system be observed over time and that patterns be learned, so that future behavior and decisions can adapt accordingly. Such adaptation can apply at many levels – from determining whether a network link to a node will be stable to predicting the future communication or computational load on the system.

5.5. Application to Other Domains

Many of the capabilities of the agile computing middleware could also be applied to other domains, such as disaster recovery. Doing so would allow a significant investment in these technologies not only from the Air Force but also the Army and the Navy to be leveraged for other beneficial purposes.

List of Acronyms

ACI – Agile Computing Infrastructure
ACK – Agile Computing Kernel
API – Application Programming Interface
CoABS – Control of Agent-based Systems
CORBA – Common Object Request Broker Architecture
COTS – Commercial Off The Shelf
CPU – Central Processing Unit
DCOM – Distributed Component Object Model
FCS – Future Combat Systems
FTP – File Transfer Protocol
MANET – Mobile AdHoc Network
MTOM – Message Transmission Optimization Mechanism
PDA – Personal Desktop Assistant
GIS – Geographic Information System
IP – Internet Protocol
JBI – Joint Battlespace Infosphere
JTRS – Joint Tactical Radio System
HMMVW – High Mobility Multipurpose Wheeled Vehicle
HTTP – HyperText Transfer Protocol
RF – Radio Frequency
RMI – Remote Method Invocation
SMTP – Simple Mail Transfer Protocol
SOA – Service Oriented Architecture
SOAP – Simple Object Access Protocol
TCP – Transmission Control Protocol
UAV – Unmanned Aerial Vehicle
UDDI – Universal Description, Discovery, and Integration
UDP – User Datagram Protocol
UGS – Unattended Ground Sensor
UGV – Unmanned Ground Vehicle
UUV – Unmanned Underwater Vehicle
VM – Virtual Machine
W3C – World Wide Web Consortium
WSDL – Web Service Description Language
XML – Extensible Markup Language

Appendix

The appendix contains technical papers that have been published in workshops and conferences that describe the results achieved by this project in greater detail. The following papers are included in the appendix:

Carvalho, M., Suri, N., Arguedas, M. (2005) *Mobile Agent-based Communications Middleware for Data Streaming in the Battlefield.* In Proceedings of the 2005 IEEE Military Communications Conference (MILCOM 2005), October 2005, Atlantic City, New Jersey.

Suri, N., Carvalho, M., Lott, J., Tortonesi, M., Bradshaw, J.M., Arguedas, M., and Breedy M. Policy-Based Bandwidth Management for Tactical Networks with the Agile Computing Middleware. In Proceedings of the 2006 IEEE Military Communications Conference (MILCOM 2006), October 2006, Washington, D.C.

Tortonesi, M., Stefanelli, C., Suri, N., Arguedas, M., and Breedy, M. Mockets: A Novel Message-oriented Communication Middleware for the Wireless Internet, in *Proceedings of International Conference on Wireless Information Networks and Systems (WINSYS 2006)*, Setúbal, Portugal, August 2006.

Stefanelli, C., Tortonesi, M., Carvalho, M., and Suri, N. Network Conditions Monitoring in the Mockets Communications Framework. In Proceedings of the 2007 IEEE Military Communications Conference (MILCOM 2007), October 2007, Orlando, FL.

Stefanelli, C., Tortonesi, M., Benvegnu, E., and Suri, N. Session Mobility in the Mockets Communication Middleware. In Proceedings of the IEEE Symposium on Computers and Communications (ISCC 2008), July 2008, Marrakech, Morocco.

Suri, N., Rebeschini, M., Breedy, M., Carvalho, M., and Arguedas, M. Resource and Service Discovery in Wireless Ad-Hoc Networks with Agile Computing. In Proceedings of the 2006 IEEE Military Communications Conference (MILCOM 2006), October 2006, Washington, D.C.

Suri, N., Rebeschini, M., Arguedas, M., Carvalho, M., Stabellini, S., and Breedy, M. Towards an Agile Computing Approach to Dynamic and Adaptive Service-Oriented Architectures. In Proceedings of the First IEEE Workshop on Autonomic Communication and Network Management (ACNM'07).

Suri, N, Marcon, M., Quitadamo, R., Rebeschini, M., Arguedas, M., Stabellini, S., Tortonesi, M., Stefanelli, C. An Adaptive and Efficient Peer-to-Peer Service-oriented Architecture fore MANET Environments with Agile Computing. In Proceedings of the Second IEEE Workshop on Autonomic Computing and Network Management (ACNM’08).